FM Function Monitor Facility User Guide

I.P.Sharp Associates

SHARP APL/370
Release Update 19.8

APL Systems Development Group
I. P. Sharp Associates Limited
2 First Canadian Place, Suite 1900
Toronto, Ontario, Canada
M5X 1E3

Telephone 416 364-5361
Telex 0622259 I P SHARP TOR

PREFACE

This book describes the functional specifications and use of the SHARP APL/370 Function Monitor Facility, □*FM*. □*FM* is issued with Release Update 19.8. For information on other enhancements released with this update, see the "Release Update 19.8 APL Programmer's Guide" (publication code: 0842 8801 E1).

Printed in Canada
January 1988
Publication Code: 0749 8711 E1

CONTENTS

OVERVIEW, 1
Specifications, 1

SETTING □*FM*, 2
Syntax, 2
Left Argument, 2
Right Argument, 3
Result, 3
Examples, 4

INQUIRING ABOUT THE MONITORING STATUS OF FUNCTIONS, 5
Syntax, 5
Left Argument, 5
Right Argument, 5
Result, 5
Example, 5

REPORTING SUMMARY MONITOR DATA, 6
Syntax, 6
Left Argument, 6
Right Argument, 6
Result, 6
Example, 7

REPORTING DETAILED MONITOR DATA, 8
Syntax, 8
Left Argument, 8
Right Argument, 8
Result, 8
Examples, 9

PROGRAMMER NOTES, 12
Notes On Recursion, 12
Error Messages, 13

OVERVIEW

⎕FM ("Function Monitor") is a system function which allows measurement and reporting of the elapsed and CPU time used during the execution of user-defined functions.

Its potential uses include:

Development workbench tool: *⎕FM* can monitor the performance of functions as they are being developed, as well as provide valuable decision criteria in the development of application programs.

Diagnostic tool: *⎕FM* provides a simple means of diagnosing problem areas in pre-existing applications for which performance has been noted to be less than optimum.

Performance tracking: *⎕FM* can easily monitor running application programs for performance problems.

Debugging and testing tool: *⎕FM* can be used to easily determine which functions, or which lines of functions have been executed.

SPECIFICATIONS

With *⎕FM*, you can specify which workspace functions are to be monitored, and the degree of monitoring detail to be performed. *⎕FM* also provides facilities for reporting and resetting monitoring statistics. *⎕FM* can record the following information:

For each monitored function:

- the number of times the function was executed
- the CPU and elapsed time spent in function execution, including and excluding sub-function calls
- the CPU and elapsed time spent in execution of non-monitored sub-functions

For each line of a monitored function:

- the number of times the line was executed
- the CPU and elapsed time spent in line execution including and excluding sub-fuction calls
- the CPU and elapsed time spent in execution of non-monitored sub-functions

⎕FM acts on the visible (most local) definition of each function. In case of an error during the execution of *⎕FM*, the state of the workspace remains unchanged.

Turning on any form of monitoring for a function turns off any previous monitor setting for that function.

SETTING □*FM*

SYNTAX:

R←α □*FM* **fnlist**

where

R - is the explicit result
α - is the monitor setting
fnlist - is an array of function names

LEFT ARGUMENT:

The left argument is an integer scalar, indicating the level of monitoring to be performed. Each individual monitor setting is a power of 2. Any combination of monitor levels can be expressed as the sum of the separate distinct monitor levels. Note that if any monitor level (other than 0) is specified, summary monitoring is automatically included with it.

0 **Turn monitoring off**
Disables monitoring for the functions named in **fnlist**, and releases any workspace storage used by previously monitored functions.

1 **Summary monitoring**
Enables recording of summary information for the functions named in **fnlist**. Summary information records one line per function, as if the function consisted of a single line. Summary monitoring includes, for each function being monitored, the number of times that function was executed, as well as the CPU and elapsed time, including and excluding both monitored and non-monitored subfunctions called, spent in that function. Summary monitoring increases workspace storage requirements by 100 bytes per function monitored.

2 **Line Execution Count monitoring**
Enables recording of the number of times each function line was executed. This monitor setting increases workspace storage by 4 bytes per function line.

4 **CPU OWN monitoring**
Enables recording of the total CPU time spent on each function line only, **excluding** any subfunctions called. This monitor level increases workspace storage by 8 bytes per line.

8 **Elapsed OWN monitoring**
Enables recording of the total elapsed time spent on each function line only, **excluding** any subfunctions called. This monitor level increases workspace storage by 8 bytes per line.

16 **CPU TOTAL monitoring**
Enables recording of the total CPU time spent on each function line, **including** any subfunctions called. This monitor level increases workspace storage by 16 bytes per line.

32 **Elapsed TOTAL monitoring**
Enables recording of the total elapsed time spent on each function line, **including** any subfunctions called. This monitor level increases workspace storage by 16 bytes per line.

Note: using monitor levels 16 and/or 32 requires an extra 4 bytes of storage. If one or the other is used, the 4 bytes will be used for storage; if both levels are used the 4 bytes is shared between them.

64 **CPU total - Non-monitored subfunctions**

Enables recording, for each function line, the total CPU time spent on non-monitored subfunctions called. This monitor level increases workspace storage by 8 bytes per function line.

128 **Elapsed total - Non-monitored subfunctions**

Enables recording, for each function line, the total elapsed time spent on non-monitored subfunctions called. This monitor level increases workspace storage by 8 bytes per function line.

RIGHT ARGUMENT:

The right argument, **fnlist**, is a character array which names the function(s) to be monitored. Each row of **fnlist** contains the name of one function.

RESULT

The explicit result is an array of shape ¯1↓⍴**fnlist**, in which each element corresponds to a row of **fnlist**. When monitoring is being turned on, the result is the level of monitoring achieved. This will be the same as the left argument, unless the left argument did not include 1 (summary) (by default, □FM includes summary with any monitor level set and therefore reflects this inclusion in the result); the function to be monitored is locked; or the name specified is not a function. When monitoring is being turned off, the result is the previous level of monitoring.

The meanings of the results are as follows:

0 - □FM was unable to turn on monitoring

1 - summary monitoring was successfully turned on or off

2 - monitoring of line execution count was successfully turned on or off

4 - monitoring of CPU own was successfully turned on or off

8 - monitoring of elapsed own was successfully turned on or off

16 - monitoring of CPU total was successfully turned on or off

32 - monitoring elapsed total was successfully turned on or off

64 - monitoring of CPU time spent in non-monitored subfunctions was successfully turned on or off

128 - monitoring of elapsed time spent in non-monitored subfunctions was successfully turned on or off

EXAMPLES:

1) Set line execution (2) and summary monitoring (1) for the two functions, *FNA* and *FNB*:

```
      3 □FM 2 3⍴'FNAFNB'
3 3
```

2) Set line execution count (2) and CPU totals (16) for function *FNC*:

```
      18 □FM 'FNC'          (summary monitor is implicitly
19                          turned on)
```

INQUIRING ABOUT THE MONITORING STATUS OF FUNCTIONS

SYNTAX:

R← ⎕*FM* **fnlist**

LEFT ARGUMENT: none

RIGHT ARGUMENT:

The right argument, **fnlist**, is a character array which names the functions under inquiry.

RESULT:

The result is an array of shape ¯1↓⍴**fnlist**, and contains values indicating the monitor level set for each function named in **fnlist**. See the description of the result in the section 'Setting ⎕*FM*' for a list of monitor settings and their meanings.

EXAMPLE:

Inquire about the monitor settings for all functions in the workspace:

```
      (⎕NL 3)⊃,⎕FM ⎕NL 3
FNA   3    summary and line execution count
FNB   3      "                 "
FNC  19    summary, line execution count, and CPU totals
FND   0    monitor turned off
```

REPORTING SUMMARY MONITOR DATA

SYNTAX:

R←¯1 □*FM* **fnlist**

LEFT ARGUMENT:

The left argument is the scalar ¯1 which indicates the report type is **summary**. Because summary monitor is implicitly turned on whenever you set any monitor level for a function, summary monitor data can be obtained for any monitored function.

RIGHT ARGUMENT:

The right argument is a character array naming the monitored function(s) for which reports are desired. *DOMAIN ERROR* results if any named function is not a monitored function.

RESULT:

The result is a matrix of shape (¯1↓ρ **fnlist**),7. Each row of the result represents the monitoring information associated with the corresponding function in the right argument. The values of each column represent the following information:

Column	Definition
0	Total number of times the function was called.
1	Total CPU seconds spent in the function, excluding any subfunction calls.
2	Total elapsed time, in seconds, spent in the function, excluding any subfunction calls.
3	Total CPU seconds spent in the function, including subfunction calls.
4	Total elapsed time, in seconds, spent in the function, including subfunction calls.
5	Total CPU seconds spent in non-monitored subfunctions.
6	Total elapsed seconds spent in non-monitored subfunctions.

Note: See the section entitled "Programmer Notes" for a discussion of the effects of monitoring recursive functions.

EXAMPLE:

To report summary monitoring information for functions *FNA*, *FNB* and *FNC*:

```
      ⍝1 □FM 3 3⍴'FNAFNBFNC'
FNA 1    0.002571   6.338891375   0.005385   13.46590363   0   0
FNB 1    0.001881   5.05624725    0.002814    7.12701225   0   0
FNC 1    0.000933   2.070765      0.000933    2.070765     0   0
```

REPORTING DETAILED MONITOR DATA

SYNTAX:

R← α □*FM* **fnlist**

LEFT ARGUMENT:

The **left argument** is a scalar representing the **negative** of any set monitor level, or a combination of levels, set on the function(s) named in **fnlist**. Reports can only be obtained for statistics specified when the monitor was turned on. For example, you can't report on elapsed time spent in non-monitored functions if □*FM* is set only to record summary information. The left argument cannot request a summary report along with a line report because a summary report uses all columns and a line report would use only the columns needed to produce the line statistics requested (see report column definitions under "Result" below).

Note: See the description of the Left Argument under "Setting □*FM*" for a list of monitor settings with their explanations.

RIGHT ARGUMENT:

The right argument is a character array naming the functions for which reports are desired. If multiple functions are specified, each must have the same number of lines or *LENGTH ERROR* will be signalled.

RESULT:

The result is an *N*×(1 to 7) matrix showing statistics requested. The number of rows depends on the number of lines in the function(s) for which data is sought. The number of columns depends on the specific data requested. The order of the columns reported by □*FM* is as listed below; however columns for statistics not requested will not be included. Columns are numbered from 0 to 6 (□*IO*←0); to find out from α what columns are reported you can execute: `((7ρ2) ⊤|α)/0 1 2 3 4 5 6`.

The meanings of all columns is as follows:

Column	Definition
0	The number of times the line or function was executed. The row corresponding to function line 0 is the number of times the function was executed.
1	Total CPU seconds spent executing the function line, excluding execution time of called subfunctions. For line 0, it represents the overhead associated with entering and exiting the function.
2	Total elapsed time, in seconds, spent executing the function line, excluding elapsed time of called subfunctions. For line 0, it represents the overhead associated with entering and exiting the function.

3 Total CPU seconds spent executing the function line, including that spent in called subfunctions.

4 Total elapsed time, in seconds, spent executing the function line, including that spent in called subfunctions.

5 Total CPU time, in seconds, spent in execution of non-monitored subfunctions called.

6 Total elapsed time, in seconds, spent in execution of non-monitored subfunctions called.

The result type is real. Precision depends on the CPU model, but is approximately 1 microsecond or less. The result is independent of the CPU multiplier.

If more than 1 function is specified in **fnlist**, and they have the same number of lines, the shape of the result will be:

(¯1↓⍴fnlist),(no. of lines),(1 to 7)

If monitoring is not turned on for the named function, or if you are requesting a report of statistics for which □*FM* was not set to monitor, *DOMAIN ERROR* is signalled. If multiple functions are named in **fnlist** and they do not all contain the same number of lines, a *LENGTH ERROR* is signalled.

Note: See the section entitled "Programmer Notes" for a discussion of the effects of monitoring recursive functions.

EXAMPLES:

1) Report monitor data for a single function *FNA*, which calls *FNB* on line [3]. □*FM* was set to monitor summary, line execution count, CPU own, elapsed own, CPU total, and elapsed total, comprising a monitor setting of 63. Summary is not included in the report request, so ¯62 is supplied as the left argument.

```
      ¯62 □FM 'FNA'
1   0.000073   0.00007    0.000073    0.00007
1   0.000764   2.079504125 0.000764   2.079504125
1   0.00077    1.232276375 0.00077    1.232276375
1   0.000864   3.027040875 0.003778  10.15405316
```

The difference between the amounts reported in columns 1 and 3 for function line 3 (the 4th row of the result) is the total CPU time spent on *FNB* (0.003778-0.000864). Similarly, the difference in columns 2 and 4 for function line 3 is the total elapsed time spent on *FNB* (10.15405316-3.027040875).

Hence, columns 3 and 4 of a summary report on *FNB* represent this difference:

```
      ¯1 □FM 'FNB'
1   0.001881  5.05624725  0.002814  7.12701225  0  0
```

You'll notice that this report has two columns more than the previous; a summary report includes all columns regardless of whether or not there are statistics to report from them. See the description of columns 5 and 6, under "Result".

2) In a clear workspace, two functions are defined both producing identical results, but using different algorithms. A monitor setting of 50 (line execution count, CPU and elapsed totals) is used for both functions and each is executed once. □FM is used to determine which function is more efficient.

```
     ∇Z←TAB1;I
[1]   I←8ρ2 ⋄ Z←⍳256
[2]   Z←⍉I⊤Z
     ∇

     ∇Z←TAB2;I
[1]   I←8ρ2 ⋄ Z←⍳256
[2]   Z←I⊤⍤1 0 Z
     ∇

       50 □FM 2 4ρ'TAB1TAB2'   (summary monitor is
51                             implicitly turned on)
       A←TAB1
       B←TAB2
       A≡B
1

       ⁻50 □FM 2 4ρ'TAB1TAB2'

1    0.0000855625  0.0000855625
1    0.0002865625  0.0002865625
1    0.0034016875  0.0034016875

1    0.00007025    0.00007025
1    0.000272      0.000272
1    0.00240825    0.00240825
```

After examining the report on the two functions, it can be seen that *TAB2* is the more efficient function, using less CPU and elapsed time overall. A report of summary information on the two functions makes this clearer:

```
       ⁻1 □FM 2 4ρ'TAB1TAB2'
1  0.003777 0.0037739375  0.003777 0.0037739375 0 0
1  0.002724 0.0027505     0.002724 0.0027505    0 0
```

3) Report line execution count, CPU and elapsed own monitor data for function *FNA*:

```
       ⁻14 □FM 'FNA'
1          0.000073    0.0007
1          0.000764    2.079504125
1          0.00077     1.232276375
1          0.000964    3.027040875
```

4) The function *TALK* is found to contain 101 lines of code. A monitor setting of 2 (line execution count) is set and the function is executed. A ¯2 report is generated and its result assigned to *I*. With the help of another defined function, *SQUEEZE*, to compress the result, the report is analyzed to determine first, which lines of the function were executed, and then which lines were not executed. See Programmer Note 11 for the definition of the *SQUEEZE* function.

```
      1⍴⍴ □CR 'TALK'
101
      2 □FM 'TALK'                (summary monitor is
3                                 implicitly turned on)
      TALK                        execute the TALK function
      I←0≠, ¯2 □FM 'TALK'
      SQUEEZE I/⍳⍴I
0-27,31-53,55-56,58-63,100  (executed lines)
      SQUEEZE (~I)/⍳⍴I
28-30,54,57,64¯99                 (unexecuted lines)
```

5) Report the same monitor data for all functions in the workspace:

```
      ¯14 □FM □NL 3
LENGTH ERROR
```

The functions do not all contain the same number of lines.

PROGRAMMER NOTES

1. Displaying or editing a function via ∇ (del editor) or fixing a function via ⎕*PDEF*, 3 ⎕*FD*, or ⎕*FX* turns off monitoring of that function.

2.)*COPY* discards the monitoring information; that is, the function, as copied, will have monitoring turned off.

3. ⎕*PACK* and ⎕*PDEF* discards the monitoring information; that is, the function, as defined, will have monitoring turned off.

4.)*SAVE* and)*LOAD* preserve monitoring information.

5. Setting monitoring on a pendent or suspended function is permitted. Monitoring will take effect immediately. However, monitoring will be performed only for those portions of lines which are executed after monitoring is turned on.

6. Values reported by ⎕*FM* will be precise even for functions which are suspended or pendent.

7. *NONCE ERROR* is signalled if the left argument of ⎕*FM* is not scalar.

8. 4 ⎕*WS* reports the additional space required by ⎕*FM*.

9. ⎕*TRAP* is charged to the function line in the environment in which the ⎕*TRAP* is executed.

10. The monitoring level of locked functions is limited to 1 (summary). Locking a function turns ⎕*FM* monitoring off; but once locked, ⎕*FM* can be turned on with a monitor level of 1.

11. Definition of the *SQUEEZE* function used in Example 4 in the section "Reporting Detailed Monitor Data":

```
      ∇ Z←SQUEEZE A;B
[1]    B←B∨1⌽B←¯1↓1,(1⌽A)≠A+1
[2]    A←⊖B/A
[3]    A[(Z=' ')/⍳⍴Z]←'-,'[⎕IO+¯1↓B/1⌽B]
      ∇
```

NOTES ON RECURSION

A recursive or re-invoked function can be monitored. The total CPU and elapsed time spent on lines including subfunctions is equal to the time spent on the root function. So, a function *FOO* calling *FOO* which, in turn calls *FOO*, may be seen as *FOO* calling *FOO*2 that calls *FOO*3, and only the data for *FOO* is recorded. Inclusion of time spent in recursion could produce misleading results, such as CPU times exceeding elapsed times, etc.

Summary monitored functions

If recursion does not occur, the column representing the total CPU seconds spent in a function, including subfunction calls, will be equal to the sum of (¯16 ⎕FM 'fnname'). If recursion does occur, the sum of (¯16 ⎕FM 'fnname') will be greater than the total CPU seconds spent in a function.

Similarly, if recursion does not occur, the column indicating total elapsed seconds spent in a function, including subfunction calls, will be equal to the sum of (¯32 ⎕FM 'fnname'). If recursion does occur, the sum of (¯32 ⎕FM 'fnname') will be greater than this result.

ERROR MESSAGES

WS FULL	There is insufficient workspace storage available to satisfy the request.
DOMAIN ERROR	The right argument is not of character type, or the left argument is not numeric, or is outside the allowed domain of ⎕FM, or a report is requested for data not collected.
NONCE ERROR	The left argument, if specified, is not a scalar.
LENGTH ERROR	The functions specified in **fnlist** do not have the same number of lines. It is reported for any settings other than ¯1 (summary).